

## **Real-time Computer Vision with Ruby**

WEDEKIND, Jan

Available from Sheffield Hallam University Research Archive (SHURA) at:

<http://shura.shu.ac.uk/3740/>

---

This document is the author deposited version. You are advised to consult the publisher's version if you wish to cite from it.

### **Published version**

WEDEKIND, Jan (2008). Real-time Computer Vision with Ruby. In: O'Reilly Open Source Convention (OSCON) 2008, Portland, Oregon.

---

### **Copyright and re-use policy**

See <http://shura.shu.ac.uk/information.html>

## O'Reilly Open Source Convention 2008

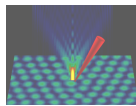


Ruby Track: Session 2471

### Real-time Computer Vision With Ruby

J. Wedekind

Wednesday, July 23rd 2008



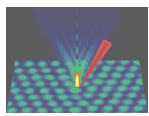
Nanorobotics EPSRC Basic Technology Grant



Microsystems and Machine Vision Laboratory



Modelling Research Centre



Brain.eval <<REQUIRED

require 'RMagick'

require 'Qt4'

require 'complex'

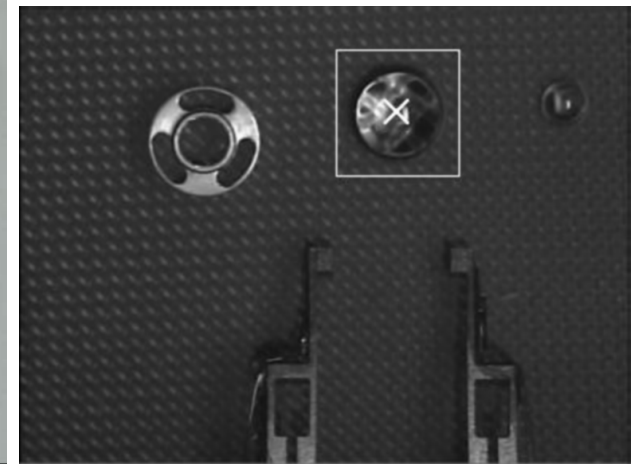
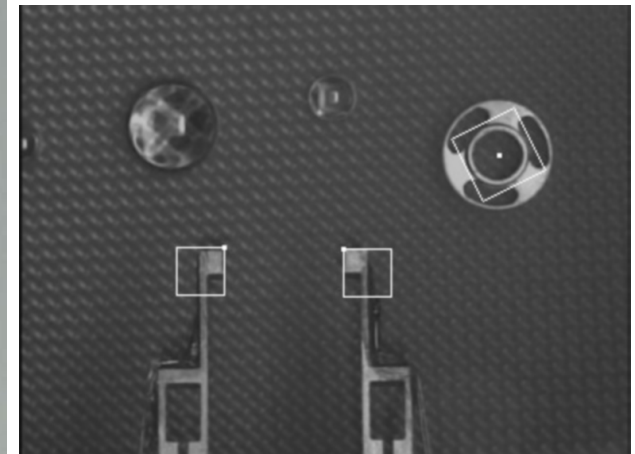
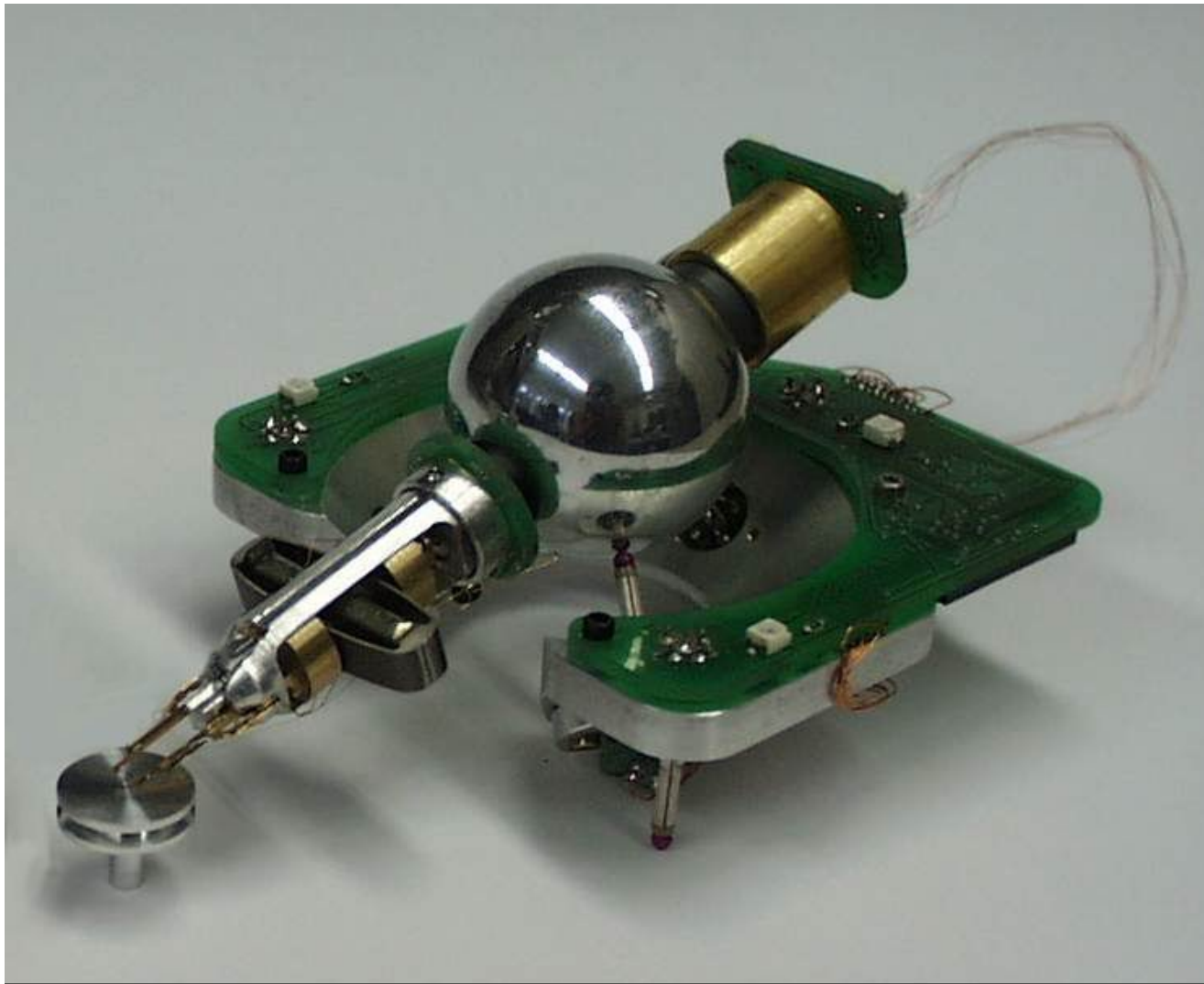
require 'matrix'

require 'narray'

REQUIRED

## Introduction

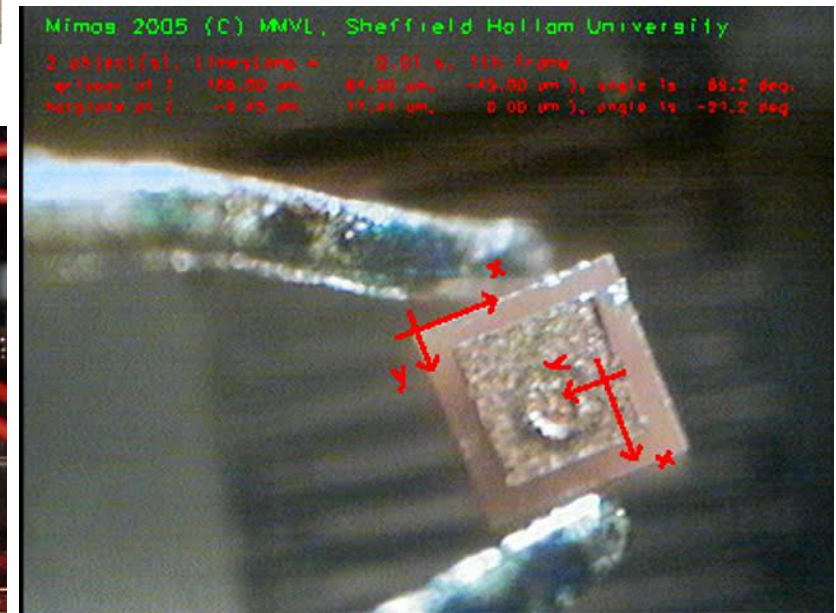
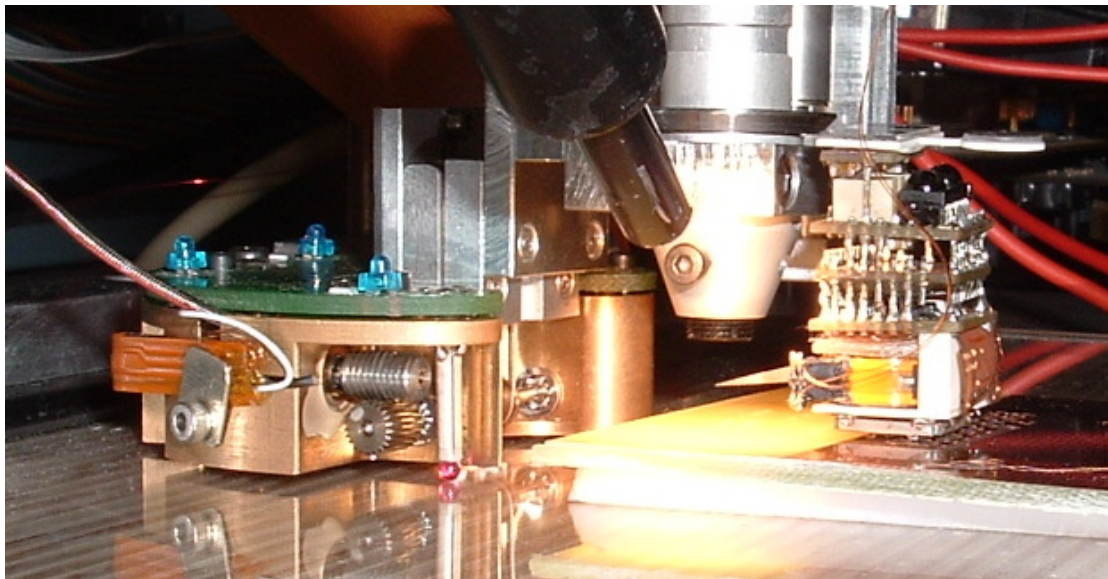
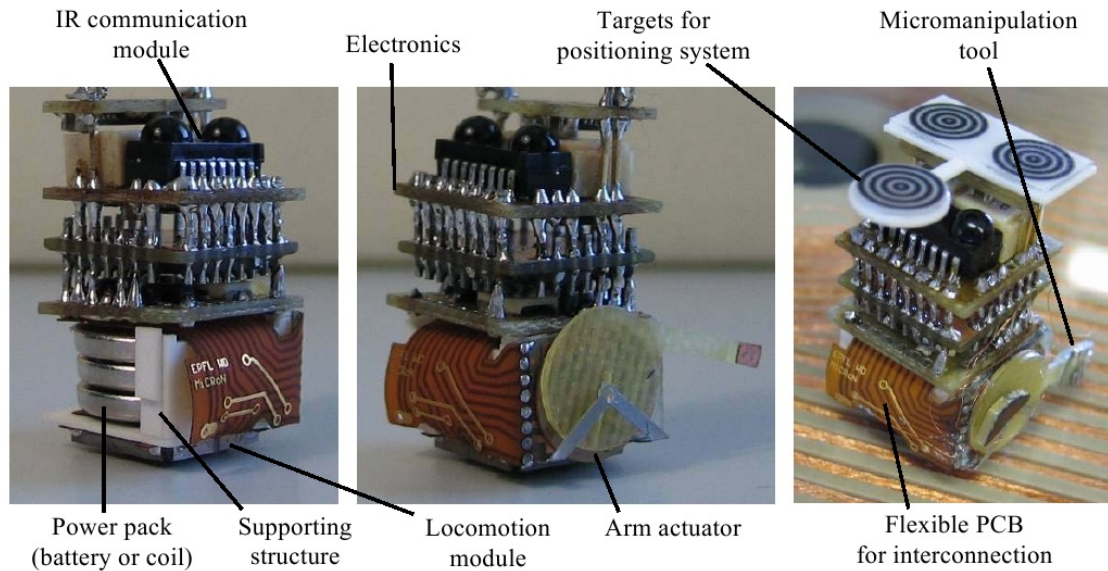
### EU Esprit MINIMAN Project



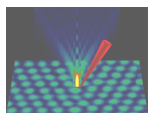


# Introduction

## EU IST MiCRoN Project

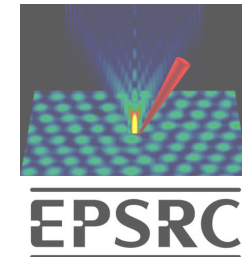
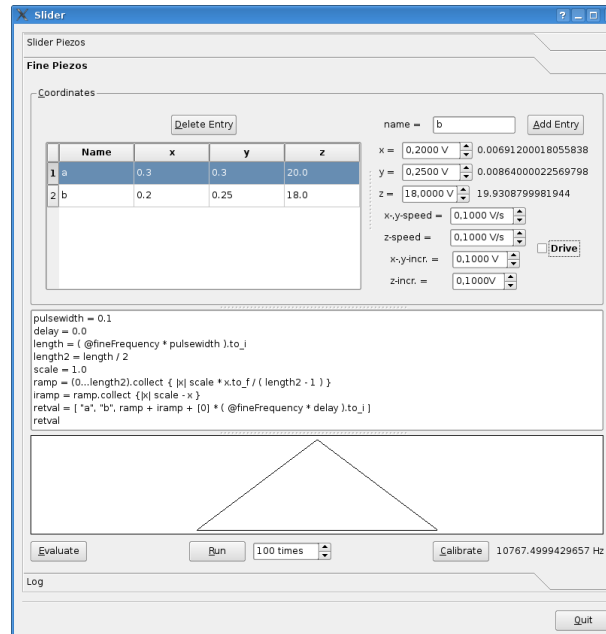
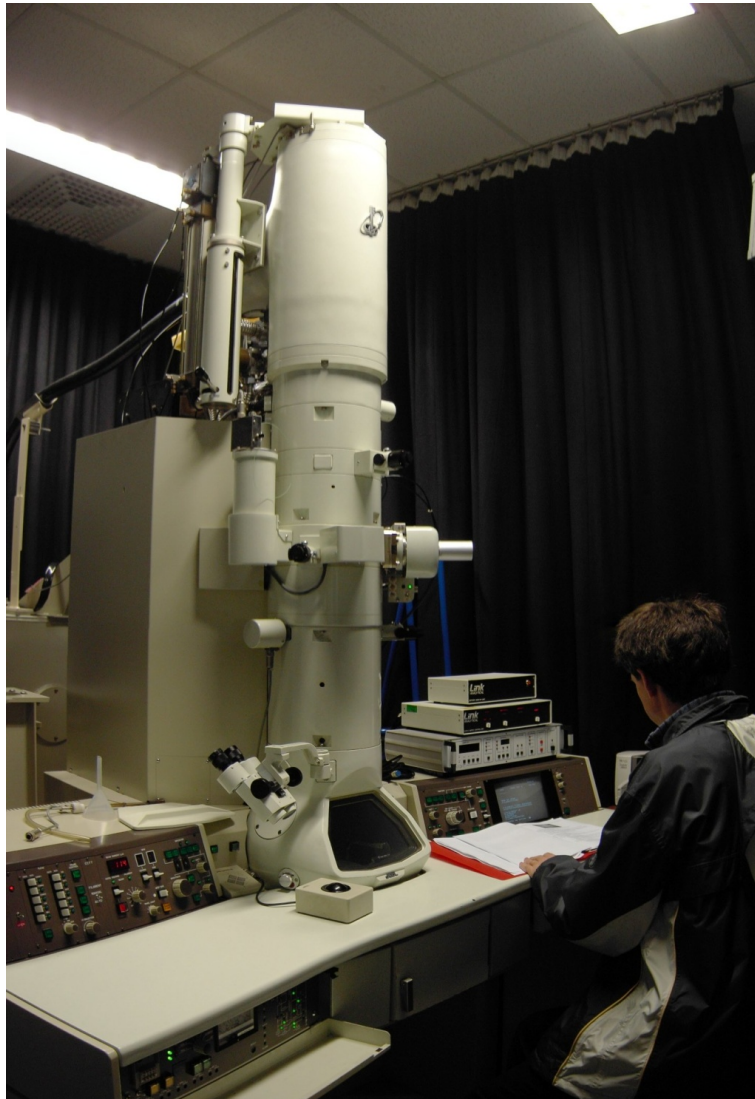






## Introduction

# EPSRC Nanorobotics Project



## Electron Microscopy

- telemanipulation
- drift-compensation
- closed-loop control

## Computer Vision

- real-time software
- system integration
- theoretical insights

## Introduction Industrial Robotics



### Default Situation

- proprietary operating system
- proprietary robot software
- proprietary process simulation software
- proprietary mathematics software
- **proprietary machine vision software**
- proprietary manufacturing software

### Total Cost of Lock-in (TCL)

- duplication of work
- integration problems
- lack of progress
- handicapped developers



## Introduction

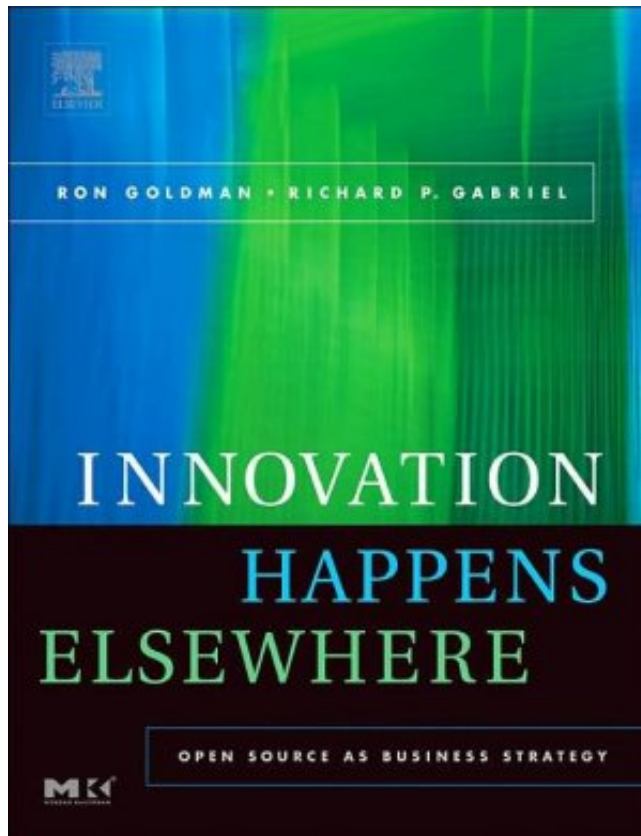
### Innovation Happens Elsewhere



#### Ron Goldman & Richard P. Gabriel

“The market need is greatest for platform products because of the importance of a reliable promise that vendor lock-in will not endanger the survival of products built or modified on the software stack above that platform.”

“It is important to remove as many barriers to collaboration as possible: social, political, and technical.”

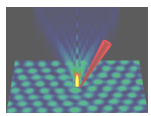


## **Design Considerations**

### **HornetsEye's Distinguishing Features**

- GPL
- Ruby
- Real-Time

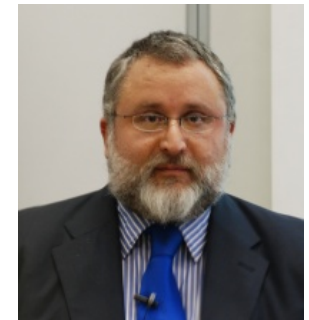




## Design Considerations GPLv3

### Four Freedoms (Richard Stallman)

1. The **freedom to run** the program, for any purpose.
2. The **freedom to study** how the program works, and adapt it to your needs.
3. The **freedom to redistribute** copies so you can help your neighbor.
4. The **freedom to improve** the program, and **release your improvements** to the public, so that the whole community benefits.



### Respect The Freedom Of Downstream Users (Richard Stallman)

GPL requires derived works to be available under the same license.

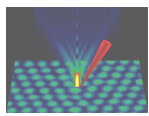
### Covenant Not To Assert Patent Claims (Eben Moglen)

GPLv3 deters users of the program from instituting patent litigation by the threat of withdrawing further rights to use the program.

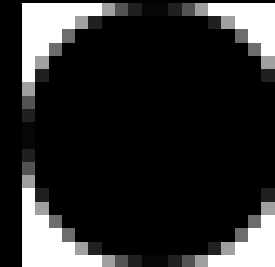
### Other (Eben Moglen)

GPLv3 has regulations against DMCA restrictions and tivoization.

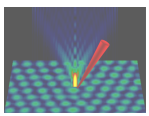




```
# -----  
img = Magick::Image.read( "circle.png" )[ 0 ]  
str = img.export_pixels_to_str( 0, 0, img.columns, img.rows, "I", Magick::CharPixel )  
arr = NArray.to_na( str, NArray::BYTE, img.columns, img.rows )  
puts ( arr / 128 ).inspect  
# NArray.byte(20,20):  
# [ [ 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1 ],  
#   [ 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 ],  
#   [ 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 ],  
#   [ 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 ],  
#   [ 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 ],  
#   [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ],  
#   [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ],  
#   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
#   [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],  
#   ...  
# -----
```

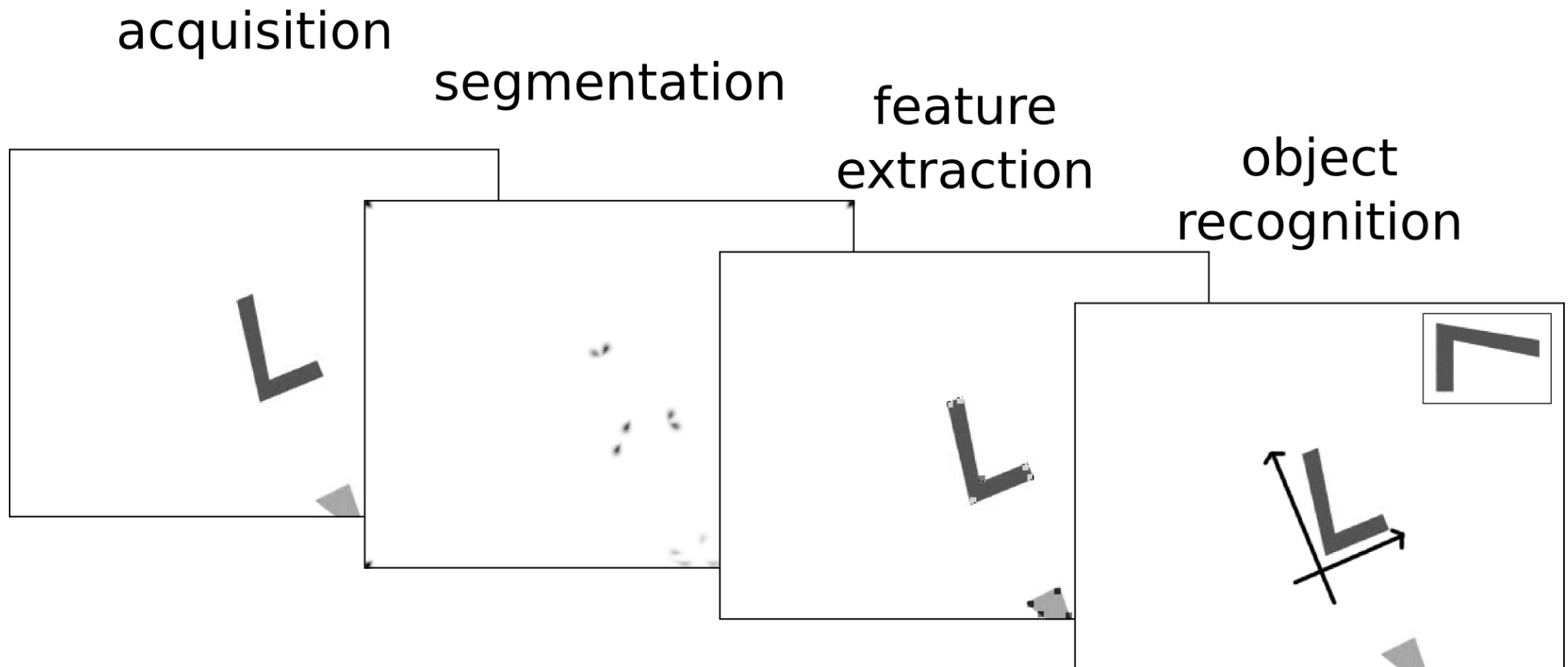


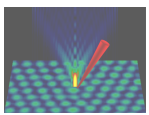
No high-level code in C++!



## Design Considerations Real-Time

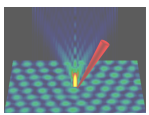
### Real-time Object Recognition





## HornetsEye Core Compact Storage

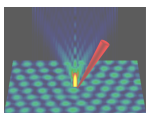
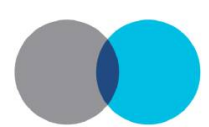
```
# -----  
class Sequence  
  Type = Struct.new( :name, :type, :size, :default, :pack, :unpack ); @@types = []  
  def Sequence.register_type( sym, type, size, default, pack, unpack )  
    eval "#{sym.to_s} = Type.new( sym.to_s, type, size, default, pack, unpack )"   
  end  
  register_type( :OBJECT, Object, 1, nil, proc { |o| [o] }, proc { |s| s[0] } )  
  register_type( :UBYTE, Fixnum, 1, 0, proc { |o| [o].pack("C") },  
    proc { |s| s.unpack("C")[0] } )  
  def initialize( type = OBJECT, n = 0, value = nil )  
    @type, @data = type, type.pack.call( value == nil ? type.default : value ) * n  
    @size = n  
  end  
  def []( i )  
    p = i * @type.size; @type.unpack.call( @data[ p...( p + @type.size ) ] )  
  end  
  def []=( i, o )  
    p = i * @type.size; @data[ p...( p + @type.size ) ] = @type.pack.call( o ); o  
  end  
end  
# -----
```



## HornetsEye Core N-Dimensional Arrays

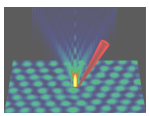
```
# -----  
class MultiArray  
  UBYTE = Sequence::UBYTE  
  OBJECT = Sequence::OBJECT  
  def initialize( type = OBJECT, *shape )  
    @shape = shape  
    stride = 1  
    @strides = shape.collect { |s| old = stride; stride *= s; old }  
    @data = Sequence.new( type, shape.inject( 1 ) { |r,d| r*d } )  
  end  
  def []( *indices )  
    @data[ indices.zip( @strides ).inject( 0 ) { |p,i| p + i[0] * i[1] } ]  
  end  
  def []=( *indices )  
    value = indices.pop  
    @data[ indices.zip( @strides ).inject( 0 ) { |p,i| p + i[0] * i[1] } ] = value  
  end  
end  
# -----
```





## HornetsEye Core Element-wise Operations

```
# -----  
class Sequence  
  attr_reader :type, :data, :size  
  def collect( type = @type )  
    retval = Sequence.new( type, @size )  
    ( 0...@size ).each { |i| retval[i] = yield self[i] }  
    retval  
  end  
end  
class MultiArray  
  attr_accessor :shape, :strides, :data  
  def MultiArray.import( type, data, *shape )  
    retval = MultiArray.new( type )  
    stride = 1; retval.strides = shape.collect { |s| old = stride; stride *= s; old }  
    retval.shape, retval.data = shape, data; retval  
  end  
  def collect( type = @data.type, &action )  
    MultiArray.import( type, @data.collect( type, &action ), *@shape )  
  end  
end  
# -----
```



## HornetsEye Core

### Return-type Coercions

```
# -----  
class Sequence  
  @@coercions = Hash.new  
  @@coercions.default = OBJECT  
  def Sequence.register_coercion( result, type1, type2 )  
    @@coercions[ [ type1, type1 ] ] = type1  
    @@coercions[ [ type2, type2 ] ] = type2  
    @@coercions[ [ type1, type2 ] ] = result  
    @@coercions[ [ type2, type1 ] ] = result  
  end  
  register_coercion( OBJECT, OBJECT, UBYTE )  
  def +( other )  
    retval = Sequence.new( @@coercions[ [ @type, other.type ] ], @size )  
    ( 0...@size ).each { |i| retval[i] = self[i] + other[i] }  
    retval  
  end  
end  
# -----
```

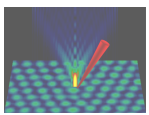
## **HornetsEye Core**

### **Fast Malloc Objects**

```
VALUE Malloc::wrapMid( VALUE rbSelf, VALUE rbOffset, VALUE rbLength )
{
    char *self; Data_Get_Struct( rbSelf, char, self );
    return rb_str_new( self + NUM2INT( rbOffset ), NUM2INT( rbLength ) );
}
```



```
m=Malloc.new(1000)
m.mid(10,4)
# "\000\000\000\000"
m.assign(10,"test")
m.mid(10,4)
# "test"
```



# HornetsEye Core Native Operations

MATERIALS AND ENGINEERING  
RESEARCH INSTITUTE

$$g, h \in \{0, 1, \dots, w-1\} \times \{0, 1, \dots, h-1\} \rightarrow \mathbb{R}$$

$$h\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = g\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) / 2$$

Ruby

$h = g / 2$

MultiArray.dfloat

Fixnum

MultiArray.dfloat( 320, 240 ):  
[ [ 245.0, 244.0, 197.0, ... ],  
[ 245.0, 247.0, 197.0, ... ],  
[ 247.0, 248.0, 187.0, ... ]  
...

MultiArray.respond\_to?( "binary\_div\_lint\_dfloat" )

no

$h = g.collect \{ |x| x / 2 \}$

yes

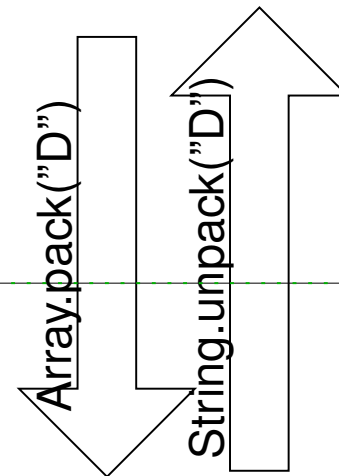
C++

```
for ( int i=0; i<n; i++ )  
    *r++ = *p++ / q;
```

MultiArray.binary\_div\_byte\_byte  
MultiArray.binary\_div\_byte\_bytergb  
MultiArray.binary\_div\_byte\_dcomplex  
MultiArray.binary\_div\_byte\_dfloat  
MultiArray.binary\_div\_byte\_dfloatrgb  
...

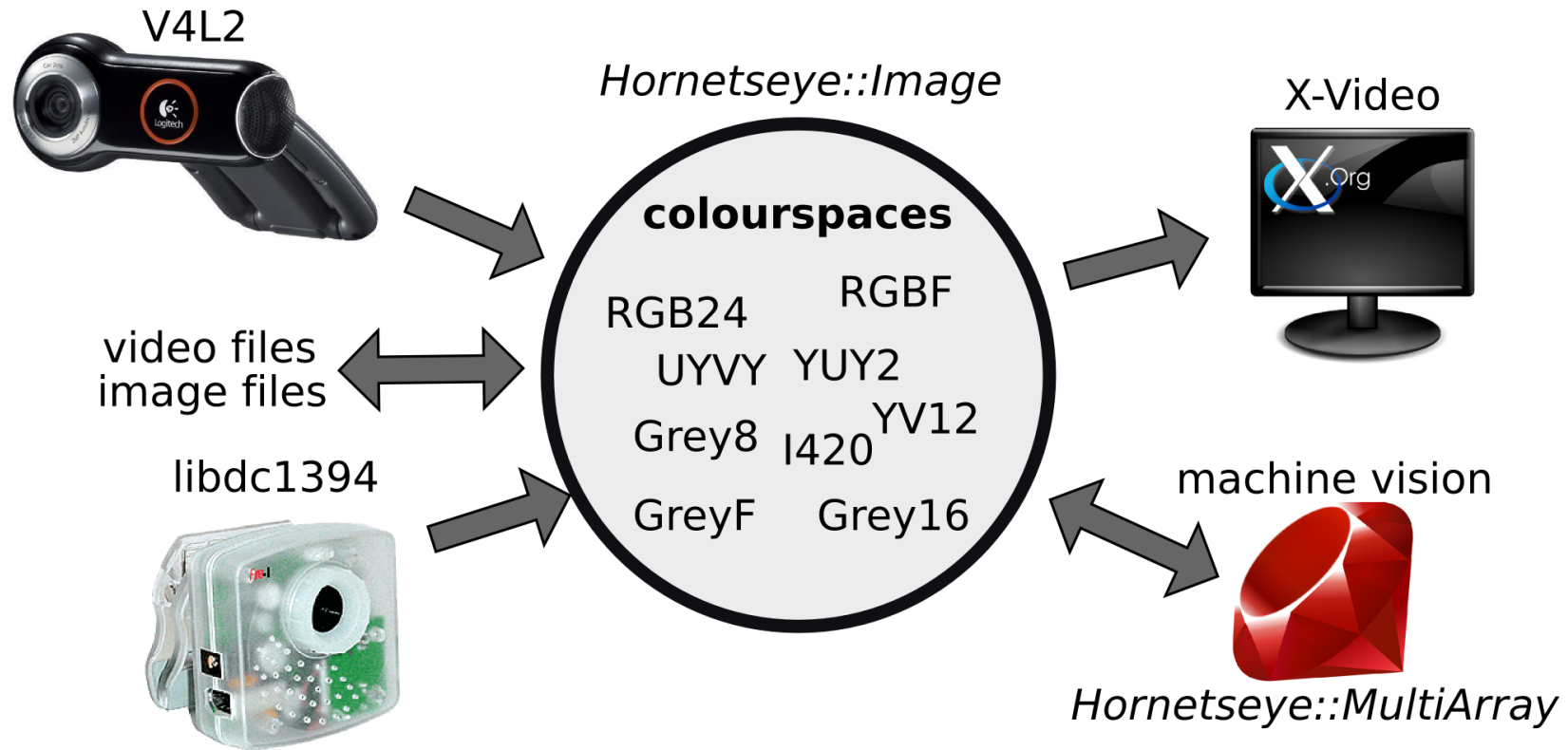
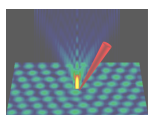
m

[3.141]



"\x54\xE3\xA5\x9B\xC4\x20\x09\x40"





$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.500 \\ 0.500 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

also see: <http://fourcc.org/>





```
mgk = Magick::Image.read( "circle.png" )[0] # code is simplified
str = magick.export_pixels_to_str( 0, 0, mgk.columns, mgk.rows, "RGB",
                                   Magick::CharPixel )
arr = MultiArray.import( MultiArray::UBYTERGB, str, mgk.columns, mgk.rows )
```



```
arr = MultiArray.load_rgb24( "circle.png" )
```

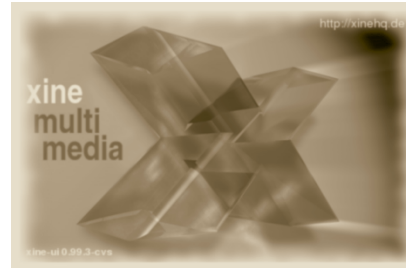
---

```
mgk = Magick::Image.new( *arr.shape ) { |x| x.depth = 8 }
mgk.import_pixels( 0, 0, arr.shape[0], arr.shape[1], "RGB", arr.to_s,
                  Magick::CharPixel )
Magick::ImageList.new.push( mgk ).write( "circle.png" )
```



```
arr = MultiArray.save_rgb24( "circle.png" )
```

## HornetsEye I/O Video Decoding



```
xine_t *m_xine = xine_new(); // code is simplified
xine_config_load( m_xine, "/home/myusername/.xine/config" );
xine_init( m_xine );
xine_video_port_t *m_videoPort = xine_new_framegrab_video_port( m_xine );
xine_stream_t *m_stream = xine_stream_new( m_xine, NULL, m_videoPort );
xine_open( m_stream, "test.avi" );
xine_video_frame_t *m_frame;
xine_get_next_video_frame( m_videoPort, &m_frame );
xine_free_video_frame( m_videoPort, &m_frame );
```



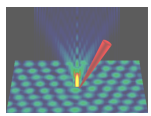
```
xine = XineInput.new( "test.avi" )
img = xine.read
```



```
// "const unsigned char *data" points to I420-data of 320x240 frame
FILE *m_control = popen( "mencoder - -o test.avi" // code is simplified
                        " -ovc lavc -lavcopts vcodec=ffv1", "w" );
fprintf( m_control, "YUV4MPEG2 W320 H240 F25000000:1000000 Ip A0:0\n" );
fprintf( m_control, "FRAME\n" );
fwrite( data, 320 * 240 * 3 / 2, 1, m_control );
```



```
# "img" is of type "HornetsEye::Image" or "HornetsEye::MultiArray"
mencoder = MEncoderOutput.new( "test.avi", 25 )
mencoder.write( img )
```



```
int m_fd = open( "/dev/video0", O_RDWR, 0 ); // code is incomplete
ioctl( VIDIOC_S_FMT, &m_format );
ioctl( VIDIOC_REQBUFS, &m_req );
ioctl( VIDIOC_QUERYBUF, &m_buf[0] );
ioctl( VIDIOC_QBUF, &m_buf[0] );
ioctl( VIDIOC_STREAMON, &type );
ioctl( VIDIOC_DQBUF, &buf )
// ...
```



```
v4l2 = V4L2Input.new
img = v4l2.read
```

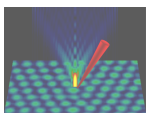


```
raw1394handle_t m_handle = dc1394_create_handle( 0 ); // code is incomplete
dc1394_cameracapture m_camera; int numCameras;
nodeid_t *m_cameraNode = dc1394_get_camera_nodes( m_handle, &numCameras, 0 );
dc1394_camera_on( m_handle, 0 );
dc1394_dma_setup_capture( m_handle, m_cameraNode[ 0 ], 0,
                          FORMAT_VGA_NONCOMPRESSED, MODE_640x480_YUV422,
                          FRAMERATE_15, 4, 1, NULL, &m_camera );
dc1394_start_iso_transmission( m_handle, m_camera.node );
// ...
```



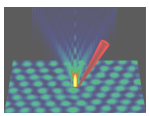
```
firewire = DC1394Input.new
img = firewire.read
```





```
# -----  
img = MultiArray.load_rgb24( "howden.jpg" )  
display = X11Display.new  
output = XImageOutput.new  
# output = OpenGLOutput.new  
window = X11Window.new( display, output,  
                          320, 240 )  
  
window.title = "Test"  
output.write( img )  
window.show  
display.eventLoop  
# -----
```





```
# -----  
xine = XineInput.new( "dvd://1" ); sleep 2  
display = X11Display.new  
output = XVideoOutput.new  
window = X11Window.new( display, output,  
                          768, 576 * 9 / 16 )  
  
window.title = "Test"  
window.show  
delay = xine.frame_duration.to_f / 90000.0  
time = Time.now  
while xine.status? and output.status?  
  output.write( xine.read )  
  time_left = delay - ( Time.now.to_f -  
                        time.to_f )  
  display.eventLoop( time_left * 1000 )  
  time += delay  
end  
# -----
```





# HornetsEye I/O

## High Dynamic Range Images

### Exposure Series



### Alignment (Hugin)



### Tonemapping (QtPfsGui)



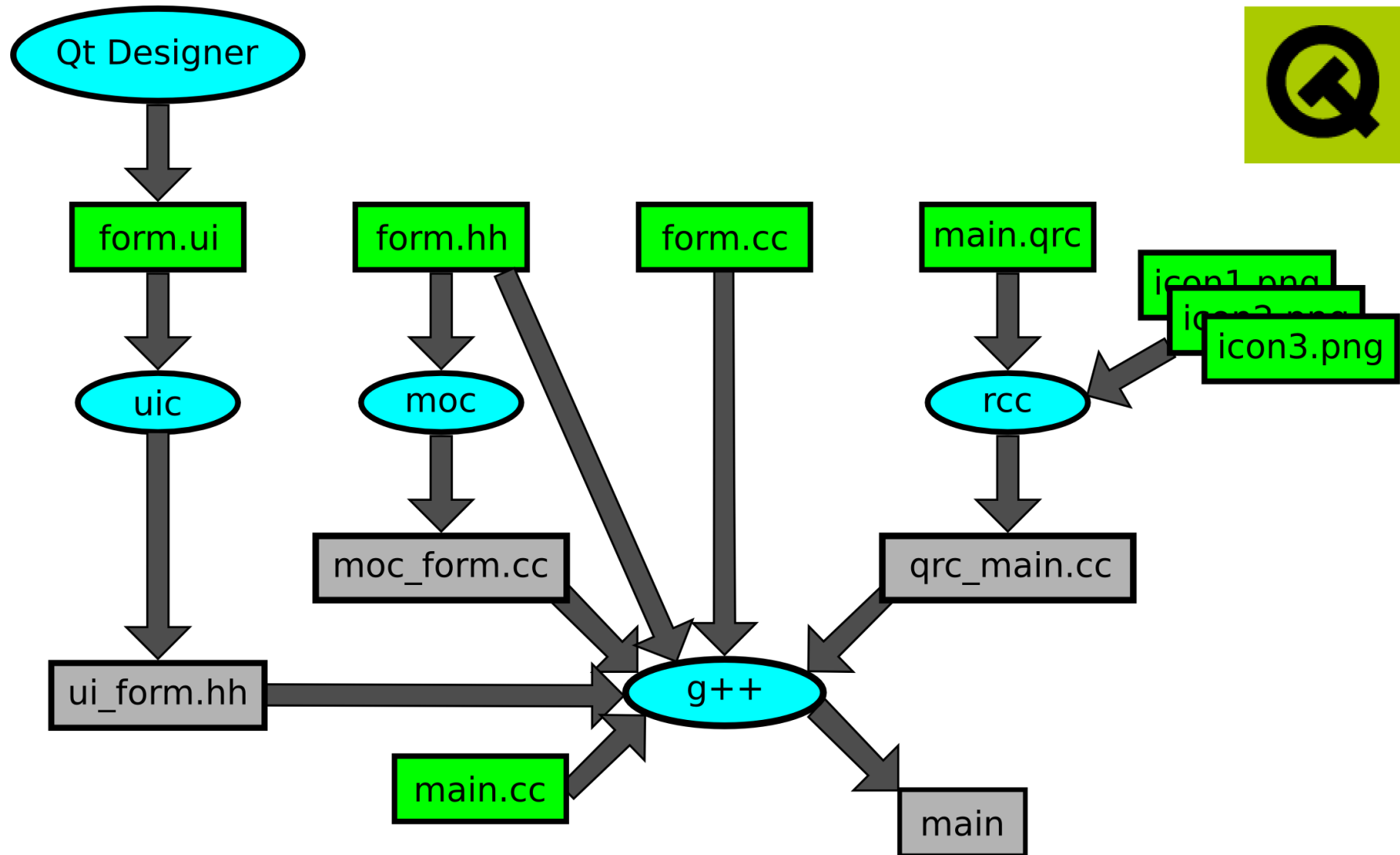
### Loading And Saving

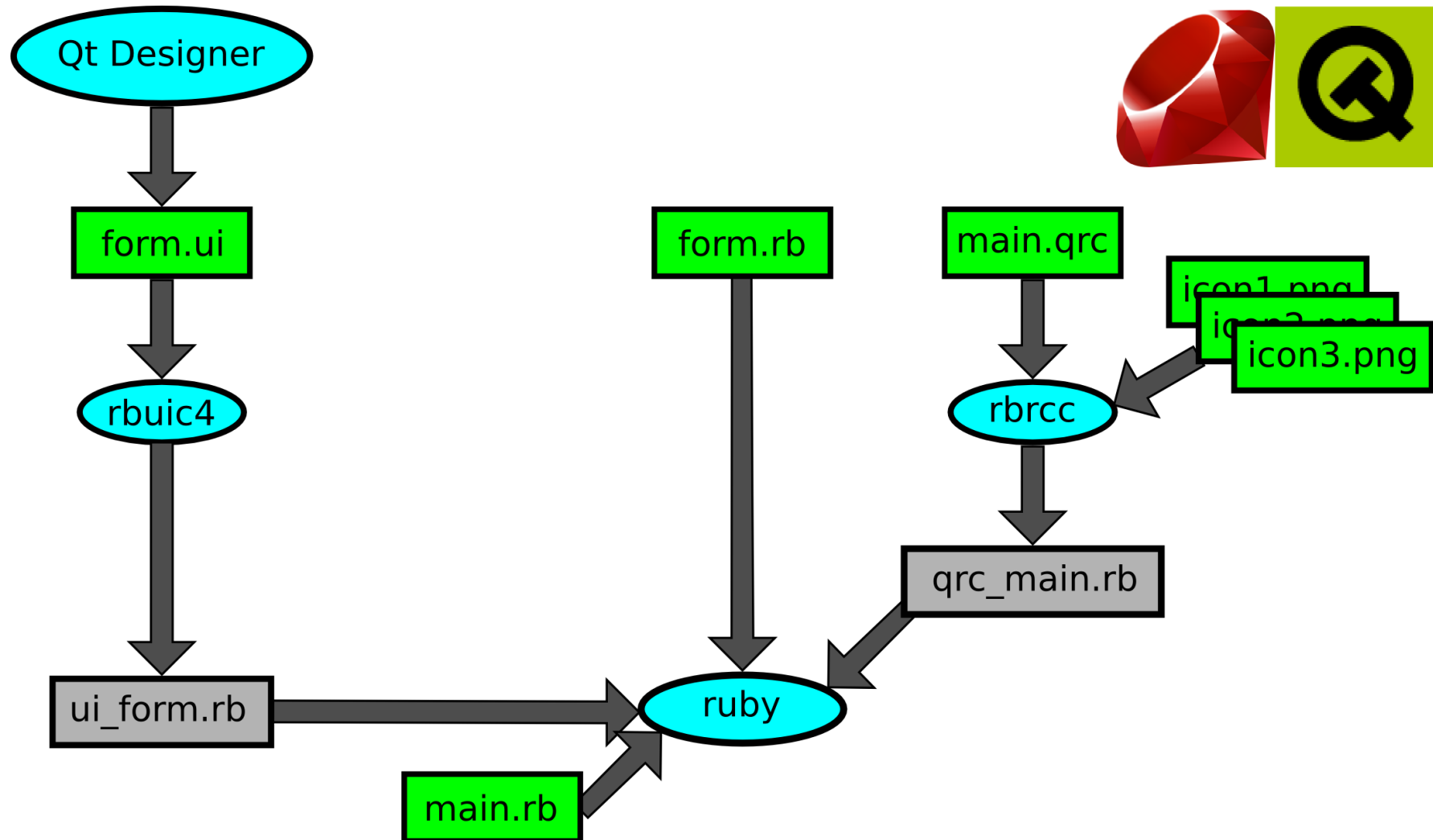


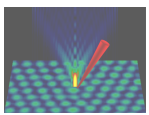
```
img = MultiArray.  
    load_rgbf("test.exr")  
img.  
    save_rgbf("test.exr")
```

# HornetsEye I/O

## Qt4-QtRuby: G++ Dataflow



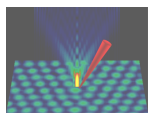
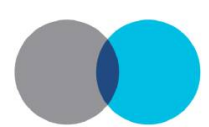




```
# -----  
class VideoPlayer < Qt::Widget  
  def initialize  
    super  
    @xvideo = Hornetseye::XvWidget.new( self )  
    layout = Qt::VBoxLayout.new( self )  
    layout.addWidget( @xvideo )  
    @xine = Hornetseye::XineInput.new( "test.avi", false )  
    @timer = startTimer( @xine.frame_duration * 1000 / 90000 )  
    resize( 640, 400 )  
  end  
  def timerEvent( e )  
    begin  
      if @xine  
        img = @xine.read  
        @xvideo.write( img )  
      end  
    rescue  
      @xine = nil  
      killTimer( @timer )  
      @xvideo.clear  
      @timer = 0  
    end  
  end  
end  
app = Qt::Application.new( ARGV )  
VideoPlayer.new.show  
app.exec  
# -----
```







## HornetsEye I/O Microsoft Windows



---

V4LInput	VFWInput
V4L2Input	DShowInput
DC1394Input	—
XineInput	—
MPlayerInput	MPlayerInput
MEncoderOutput	MEncoderOutput
X11Display	W32Display
X11Window	W32Window
XImageOutput	GDIOutput
OpenGLOutput	—
XVideoOutput	—

---

## Introduction From Here On

Brain.eval <<REQUIRED  
require 'hornetseye'  
include Hornetseye  
REQUIRED

# Computer Vision With Ruby

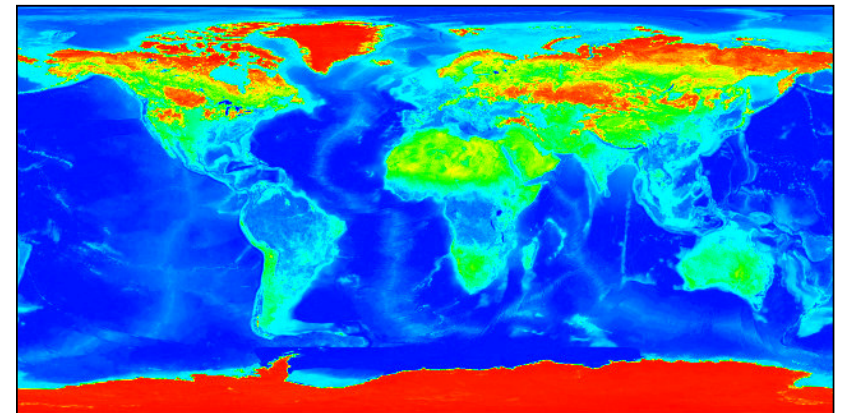
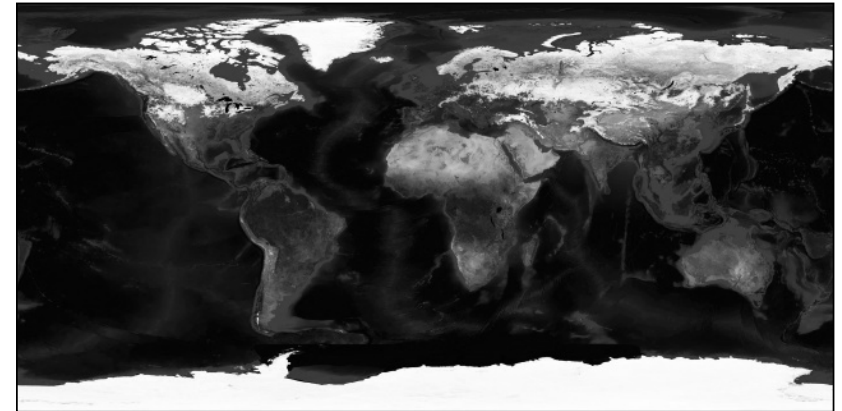
## Look-Up-Tables (LUTs)

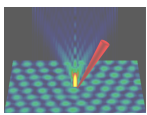
$$g \in \{0, 1, \dots, w\} \times \{0, 1, \dots, h\} \rightarrow \{0, 1, \dots, 255\}$$

$$m \in \{0, 1, \dots, 255\} \rightarrow \{0, 1, \dots, 255\}^3$$

$$h\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = m\left(g\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right)\right)$$

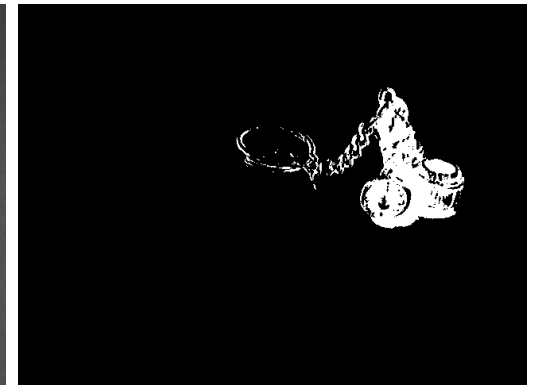
```
# -----
img = MultiArray.load_grey8( "test.jpg" )
class Numeric
  def clip( range )
    [ [ self, range.begin ].max, range.end ].min
  end
end
colours = {}
for i in 0...256
  hue = 240 - i * 240.0 / 256.0
  colours[i] =
    RGB( ( ( hue - 180 ).abs - 60 ).clip( 0...60 ) * 255 / 60.0,
          ( 120 - ( hue - 120 ).abs ).clip( 0...60 ) * 255 / 60.0,
          ( 120 - ( hue - 240 ).abs ).clip( 0...60 ) * 255 / 60.0 )
end
img.map( colours, MultiArray::UBYTE, 256 ).display
# -----
```





```
# -----
class MultiArray
  def MultiArray.ramp1( *shape )
    retval = MultiArray.new( MultiArray::LINT, *shape )
    for x in 0...shape[0]
      retval[ x, 0...shape[1] ] = x
    end
    retval
  end
  # def MultiArray.ramp2 ...
end
input = V4LInput.new
x, y =
  MultiArray.ramp1( input.width, input.height ),
  MultiArray.ramp2( input.width, input.height )
display = X11Display.new
output = XVideoOutput.new
window = X11Window.new( display, output, 640, 480 )
window.title = "Thresholding"
window.show
while input.status? and output.status?
  img = input.read_grey8
  mask = img.binarise_lt( 48 )
  result = ( img / 4 ) * ( mask + 1 )
  if mask.sum > 0
    bbox = [ x.mask( mask ).range, y.mask( mask ).range ]
    result[ *bbox ] *= 2
  end
  output.write( result )
  display.processEvents
end
# -----
```

### Compute Bounding Box



0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5

2	3	4	2	3	1	2	3	2	3
---	---	---	---	---	---	---	---	---	---

$$1 \leq x \leq 4$$





# Computer Vision With Ruby

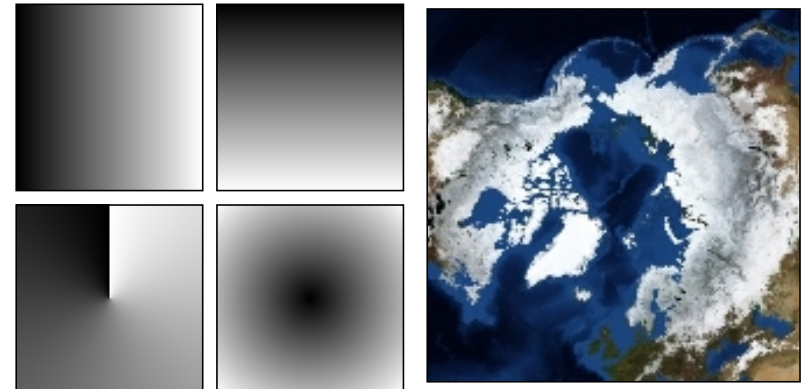
## Warps (Use Image As LUT)

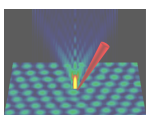
$$\begin{aligned}
 g &\in \{0, 1, \dots, w-1\} \times \{0, 1, \dots, h-1\} \rightarrow \mathbb{R}^3 \\
 h &\in \{0, 1, \dots, w'-1\} \times \{0, 1, \dots, h'-1\} \rightarrow \mathbb{R}^3 \\
 W &\in \{0, 1, \dots, w'-1\} \times \{0, 1, \dots, h'-1\} \rightarrow \mathbb{Z}^2
 \end{aligned}
 \quad
 h\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{cases} g(W(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix})) & \text{if } W(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) \in \{0, 1, \dots, w-1\} \times \{0, 1, \dots, h-1\} \\
 0 & \text{otherwise}
 \end{cases}$$

```

# -----
class MultiArray
# def MultiArray.ramp1 ...
def MultiArray.ramp2( *shape )
  retval = MultiArray.new( MultiArray::LINT, *shape )
  for y in 0...shape[1]
    retval[ 0...shape[0], y ] = y
  end
  retval
end
end
img = MultiArray.load_rgb24( "test.jpg" )
w, h = *img.shape; c = 0.5 * h
x, y = MultiArray.ramp1( h, h ), MultiArray.ramp2( h, h )
warp = MultiArray.new( MultiArray::LINT, h, h, 2 )
warp[ 0...h, 0...h, 0 ], warp[ 0...h, 0...h, 1 ] =
  ( ( ( x - c ).atan2( y - c ) / Math::PI + 1 ) * w / 2 - 0.5 ),
  ( ( x - c ) ** 2 + ( y - c ) ** 2 ).sqrt
img.warp_clipped( warp ).display
# -----

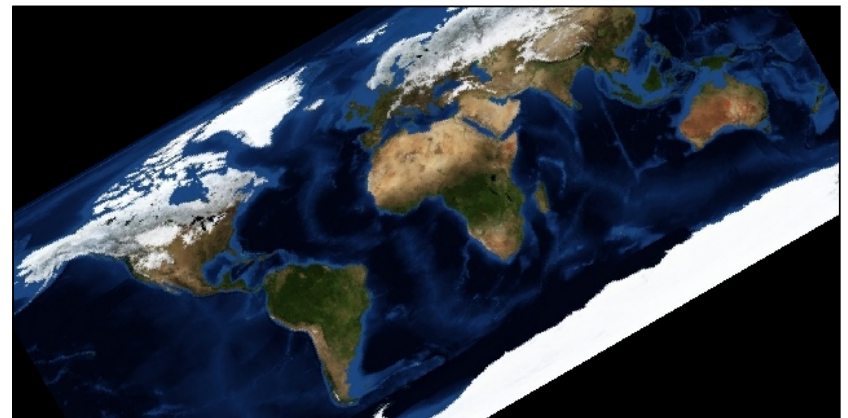
```



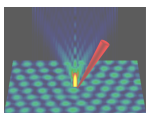


```
# -----  
class MultiArray  
  def MultiArray.ramp1( *shape )  
    retval = MultiArray.new( MultiArray::LINT, *shape )  
    for x in 0...shape[0]  
      retval[ x, 0...shape[1] ] = x  
    end  
    retval  
  end  
  # def MultiArray.ramp2 ...  
end  
img = MultiArray.load_rgb24( "test.jpg" )  
w, h = *img.shape  
v = Vector[ MultiArray.ramp1( w, h ) - w / 2,  
             MultiArray.ramp2( w, h ) - h / 2 ]  
angle = 30.0 * Math::PI / 180.0  
m = Matrix[ [ Math::cos( angle ), -Math::sin( angle ) ],  
             [ Math::sin( angle ), Math::cos( angle ) ] ]  
warp = MultiArray.new( MultiArray::LINT, w, h, 2 )  
warp[ 0...w, 0...h, 0 ], warp[ 0...w, 0...h, 1 ] =  
  ( m * v )[0] + w / 2, ( m * v )[1] + h / 2  
img.warp_clipped( warp ).display  
# -----
```

$$W_{\alpha} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$







# Computer Vision With Ruby

## Center Of Gravity And Principal Components



## Computer Vision With Ruby Linear Shift-Invariant Filters

**Input Image**



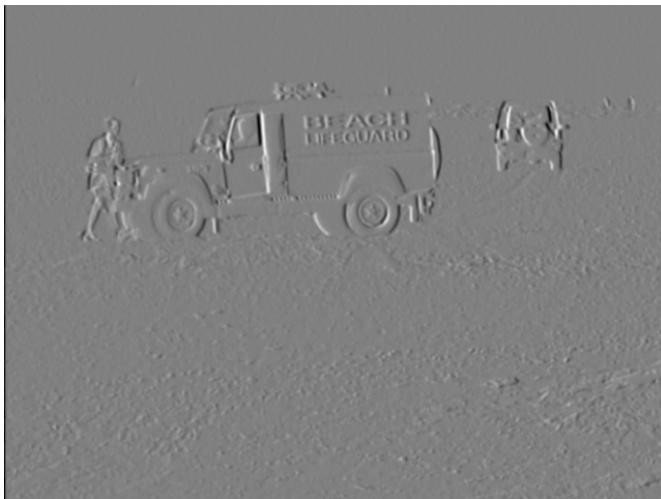
**Sharpen**



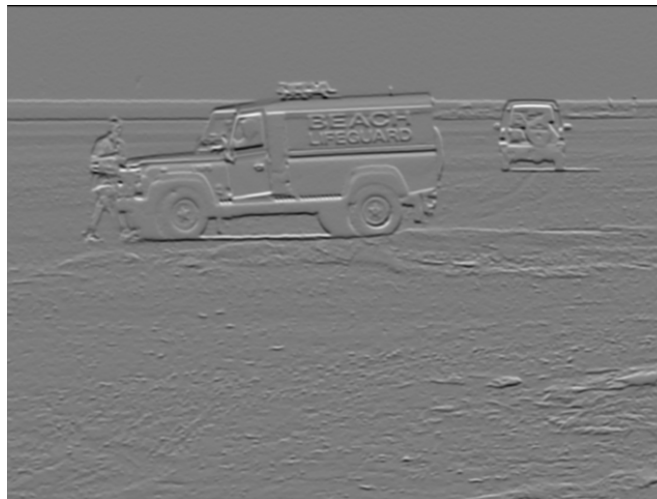
**Gaussian Blur**



**Gauss-Gradient (X)**



**Gauss-Gradient (Y)**





# Computer Vision With Ruby

## Edge- And Corner-Images

**Input Image**



**Sobel**



**Gauss-Gradient**



**Harris-Stephens**



**Kanade-Lucas-Tomasi**



# Computer Vision With Ruby

## (Inverse Compositional) Lucas-Kanade

**given:** template  $T$ , image  $I$ , previous pose  $\vec{p}$

**sought:** pose-change  $\Delta\vec{p}$

$$\operatorname{argmin}_{\Delta\vec{p}} \int_{\vec{x} \in T} \|T(\vec{x}) - I(W_{\vec{p}}^{-1}(W_{\Delta\vec{p}}^{-1}(\vec{x})))\|^2 d\vec{x} = (*)$$

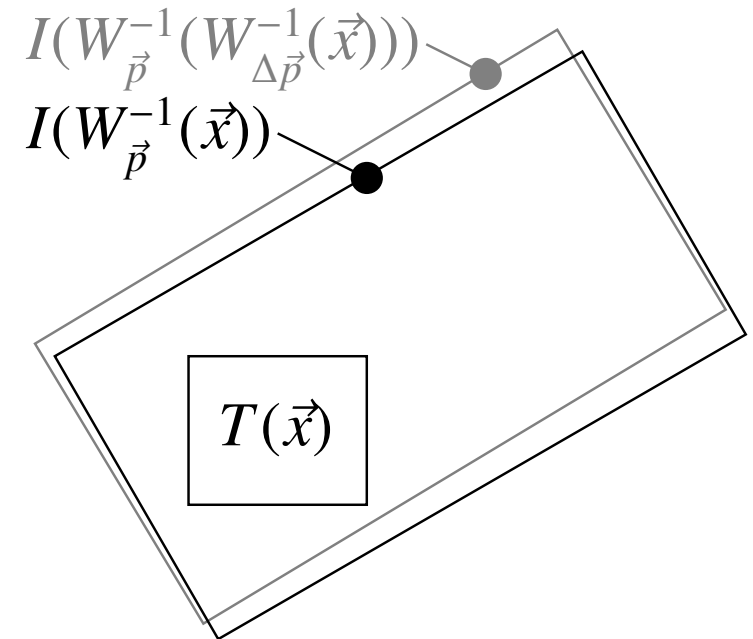
$$(1) T(\vec{x}) - I(W_{\vec{p}}^{-1}(W_{\Delta\vec{p}}^{-1}(\vec{x}))) = T(W_{\Delta\vec{p}}(\vec{x})) - I(W_{\vec{p}}^{-1}(\vec{x}))$$

$$(2) T(W_{\Delta\vec{p}}(\vec{x})) \approx T(\vec{x}) + \left( \frac{\delta T}{\delta \vec{x}}(\vec{x}) \right)^T \cdot \left( \frac{\delta W_{\vec{p}}}{\delta \vec{p}}(\vec{x}) \right) \cdot \Delta\vec{p}$$

$$(*) \stackrel{(1,2)}{=} \operatorname{argmin}_{\vec{p}} (\|\mathcal{H} \vec{p} + \vec{b}\|^2) = (\mathcal{H}^T \mathcal{H})^{-1} \mathcal{H}^T \vec{b}$$

$$\text{where } \mathcal{H} = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots \\ h_{2,1} & h_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \text{ and } \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \end{pmatrix}$$

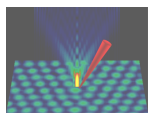
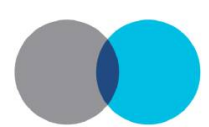
$$h_{i,j} = \left( \frac{\delta T}{\delta \vec{x}}(\vec{x}_i) \right)^T \cdot \left( \frac{\delta W_{\vec{p}}}{\delta p_j}(\vec{x}_i) \right), b_i = T(\vec{x}_i) - I(W_{\vec{p}}^{-1}(\vec{x}_i))$$



S. Baker and I. Matthew: "Lucas-Kanade 20 years on: a unifying framework"

[http://www.ri.cmu.edu/projects/project\\_515.html](http://www.ri.cmu.edu/projects/project_515.html)



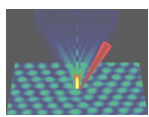


## Initialisation

```
p = Vector[ xshift, yshift, rotation ]  
w, h, sigma = tpl.shape[0], tpl.shape[1], 5.0  
x, y = xrand( w, h ), yrand( w, h )  
gx = tpl.gauss_gradient_x( sigma )  
gy = tpl.gauss_gradient_y( sigma )  
c = Matrix[ [ 1, 0 ], [ 0, 1 ], [ -y, x ] ] * Vector[ gx, gy ]  
hs = ( c * c.covector ).collect { |e| e.sum }
```

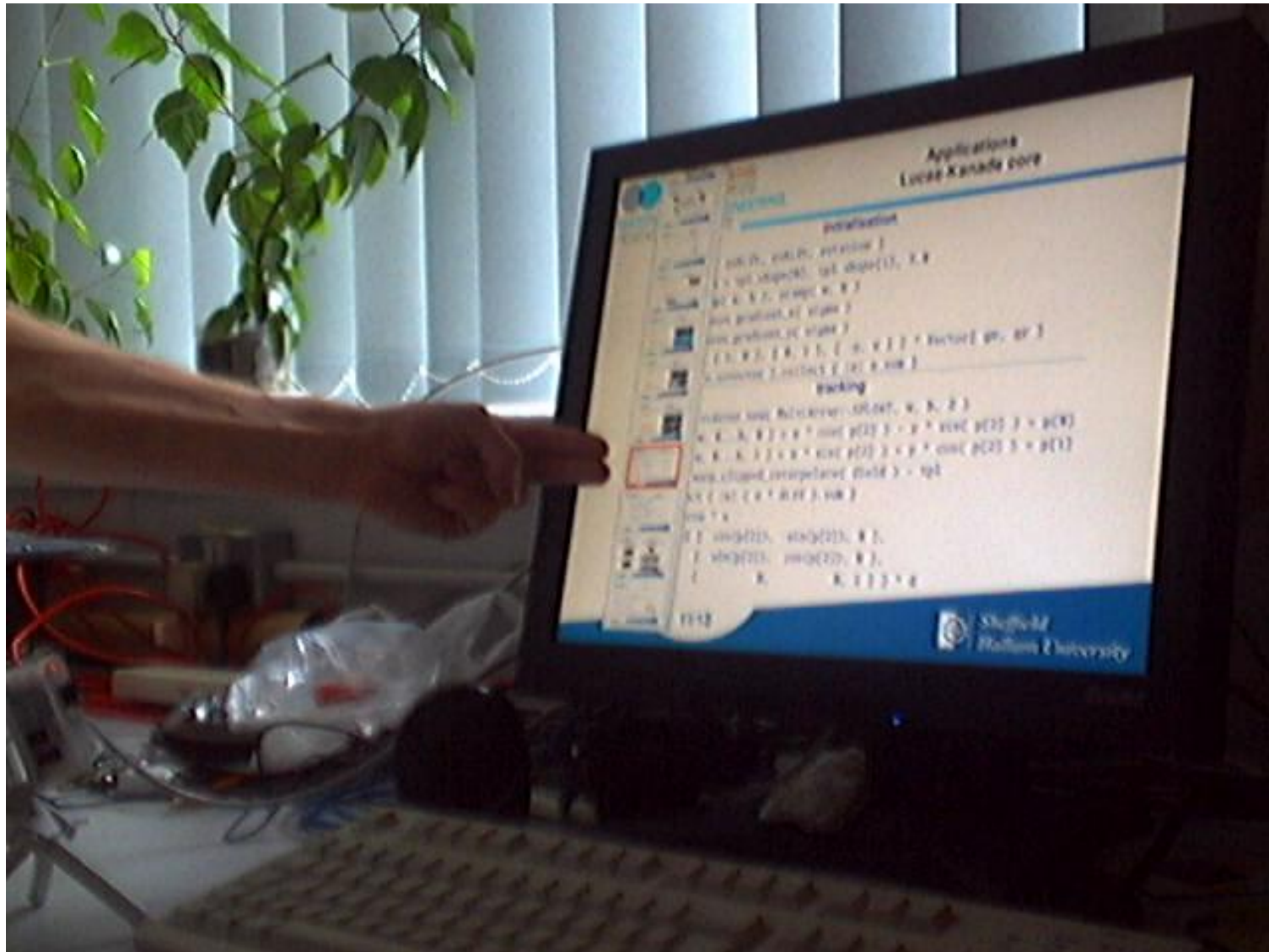
## Tracking

```
field = MultiArray.new( MultiArray::SFLOAT, w, h, 2 )  
field[ 0...w, 0...h, 0 ] = x * cos( p[2] ) - y * sin( p[2] ) + p[0]  
field[ 0...w, 0...h, 1 ] = x * sin( p[2] ) + y * cos( p[2] ) + p[1]  
diff = img.warp_clipped_interpolate( field ) - tpl  
s = c.collect { |e| ( e * diff ).sum }  
d = hs.inverse * s  
p += Matrix[ [ cos(p[2]), -sin(p[2]), 0 ],  
              [ sin(p[2]), cos(p[2]), 0 ],  
              [ 0, 0, 1 ] ] * d
```



# Computer Vision With Ruby Interactive Presentation Software

MATERIALS AND ENGINEERING  
RESEARCH INSTITUTE



## Current/Future Work

- feature extraction
  - multiresolution Lucas-Kanade
  - wavelet-based features
- feature descriptors
  - appearance templates
- feature based object recognition
  - geometric hashing
  - RANSAC
- feature based tracking
  - bounded hough transform
- parallel processing

No high-level code in C++!



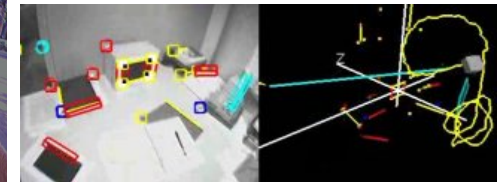
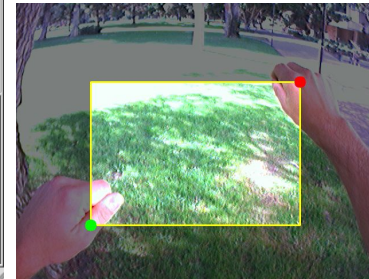
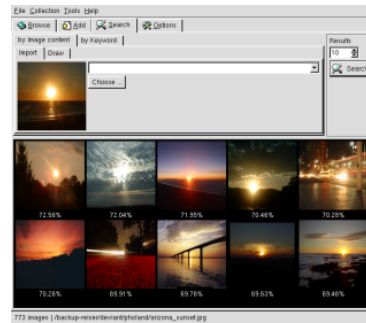
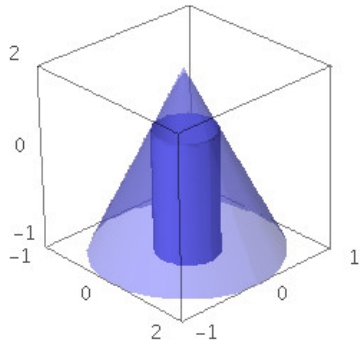
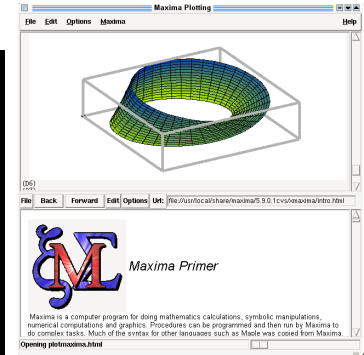
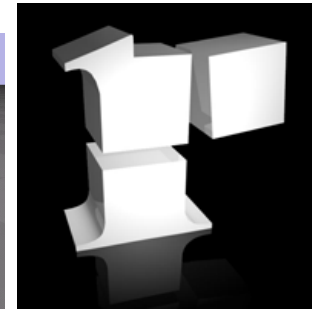
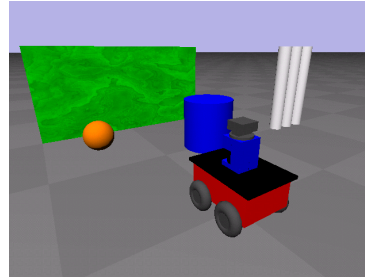
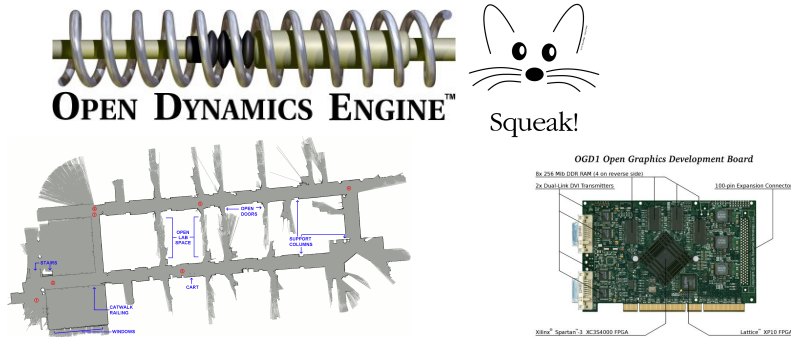
## Appeal

Computer vision only will happen if we ...

- break with business as usual
- remove all barriers to collaboration
- allow users and developers to innovate
- need fully hackable hardware
- fight for a free software stack



Let's do it!



**STRONGTALK**

